(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) **EP 4 386 596 A1**

(12) **EUROPEAN PATENT APPLICATION**
published in accordance with Art. 153(4) EPC

(43) Date of publication:
19.06.2024 Bulletin 2024/25

(21) Application number: **21931940.7**

(22) Date of filing: **08.09.2021**

(51) International Patent Classification (IPC):
G06F 21/56 $^{(2013.01)}$ ***G06F 21/53*** $^{(2013.01)}$

(52) Cooperative Patent Classification (CPC):
G06F 21/566; G06F 21/53; G06F 21/56

(86) International application number:
PCT/KR2021/012194

(87) International publication number:
WO 2023/027228 (02.03.2023 Gazette 2023/09)

(84) Designated Contracting States:
AL AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HR HU IE IS IT LI LT LU LV MC MK MT NL NO PL PT RO RS SE SI SK SM TR
Designated Extension States:
BA ME
Designated Validation States:
KH MA MD TN

(30) Priority: **26.08.2021 KR 20210112880**

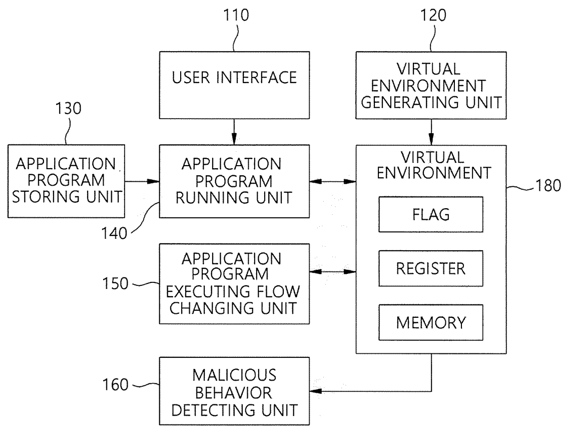
(71) Applicant: **Seculetter Co., Ltd.**
Sujeong-gu
Seongnam-si, Gyeonggi-do 13453 (KR)

(72) Inventors:
- **LIM, Cha Sung**
Yongin-si
Gyeonggi-do 16824 (KR)
- **YANG, Seung Hwan**
Anyang-si
Gyeonggi-do 14046 (KR)

(74) Representative: **Murgitroyd & Company**
165-169 Scotland Street
Glasgow G5 8PL (GB)

(54) METHOD AND DEVICE FOR DETECTING MALIGNANCY OF NON-PORTABLE EXECUTABLE FILE THROUGH EXECUTION FLOW CHANGE OF APPLICATION PROGRAM

(57) A method for detecting a maliciousness of a non-portable executable file according to the present invention includes the steps of: executing a non-portable executable file by running an application program corresponding to the non-portable executable file in a virtual environment; monitoring the execution of the application program; breaking the execution of the application program at a predetermined breakpoint during the monitoring of the execution of the application program; changing an executing flow of the application program in a breaking state of the execution of the application program and resuming the execution of the application program; and detecting a malicious behavior executed after resuming the execution of the application program.


110
120
USER INTERFACE
VIRTUAL ENVIRONMENT GENERATING UNIT
130
APPLICATION PROGRAM STORING UNIT
APPLICATION PROGRAM RUNNING UNIT
VIRTUAL ENVIRONMENT
180
FLAG
140
APPLICATION PROGRAM EXECUTING FLOW CHANGING UNIT
150
REGISTER
MEMORY
160
MALICIOUS BEHAVIOR DETECTING UNIT

EP 4 386 596 A1

Description

[Technical Field]

[0001] The present invention relates to a method and an apparatus for detecting maliciousness of a non-portable executable file, and more particularly, to a method and an apparatus for detecting maliciousness of a non-portable executable file by a behavior based inspection method.

[Background Art]

[0002] With the widespread use of the Internet and wireless communication devices, the transmission routes of malicious software or malicious codes are diversifying, and the degree of damage caused by the malicious software or malicious codes is increasing every year. The malicious code refers to software which is intentionally designed to perform malicious activities such as destroying a system or leaking information against the will and the interests of the user. Types of the malicious codes include hacking tools such as virus, worm, Trojan, backdoor, logic bomb, or trap door, malicious spyware, and ad-ware. The malicious code causes various problems such as leakage of personal information such as user identification information (ID) and a password, target system control, file deletion/change, system destruction, service denial of application program/system, core data leakage, or other hacking program installation through a self-reproduction function or automatic propagation function, and the damage is very diverse and serious.

[0003] The advanced persistent threat (APT) attack, which has become a hot topic in recent years, continuously utilizes various types of malicious codes by applying a high level of attack techniques to allow an attacker to set a specific target and extract the targeted information. Specifically, in many cases, the APT attack is not detected in an initial invasion stage and non-portable executable (Non-PE) files including mainly malicious codes are widely used. This is because a program (for example, document creating program or image program) for executing the non-portable executable file basically has a certain level of security vulnerability and when the malicious code is included in the non-portable executable file, a variant malicious code may be easily generated according to the file change. Here, the "non-portable executable file" is a concept opposed to an portable executable (PE) file or an executable file and refers to a file which is not executed by itself. For example, the non-portable executable file may be document files such as a PDF file, a HWP file, a word file, and an excel file, image files such as a JPG file, video files, java script files, or HTML files.

[0004] As a method for inspecting the maliciousness of the non-portable executable file according to the related art, there is a signature based inspection method. This method is a method for inspecting whether the non-portable executable file has a signature of a malicious code. However, most malicious non-portable executable file includes the malicious code in a script such as a java script or a macro script or in some cases, encodes the script to avoid the diagnosis so that it is difficult to know which script exists in the non-portable executable file itself. Accordingly, it is almost impossible to appropriately inspect whether the non-portable executable file is malicious using the signature based inspection method of the related art.

[0005] As another method for inspecting whether the non-portable executable file is malicious, there is a behavior based inspection method. This method is a method which inspects whether the malicious behavior occurs by actually executing the non-portable executable file. However, the malicious code included in the non-portable executable file may be designed such that when a specific condition (for example, a version of the application program or an operating system environment) is not satisfied, no behavior occurs. Accordingly, in the case of the behavior based inspection method, the non-portable executable file needs to be executed in all versions of the application program and various operating system environments and the behavior needs to be observed. Therefore, it takes a long time to analyze and it is difficult to accurately determine whether it is malicious.

[Disclosure]

[Technical Problem]

[0006] A technical object to be achieved by the present invention is to provide a method and an apparatus for detecting maliciousness of a non-portable executable file which efficiently detect maliciousness of a non-portable executable file designed such that when a specific condition, such as a version of an application program or an operating system environment, is not satisfied, a malicious behavior does not occur.

[0007] The technical object to be achieved by the present invention is not limited to the above-mentioned technical objects, and other technical objects, which are not mentioned above, can be clearly understood by those skilled in the art from the following descriptions.

[Technical Solution]

[0008] In order to achieve the technical object, according to an aspect of the present invention, a method for detecting maliciousness of a non-portable executable file includes the steps of: executing a non-portable executable file by running an application program corresponding to the non-portable executable file in a virtual environment; monitoring the execution of the application program; breaking the execution of the application program at a predetermined breakpoint during the monitoring of the execution of the application program; changing an executing flow of the application program in a breaking state of the execution of the application program and resuming the execution of the application program; and detecting a malicious behavior executed after resuming the execution of the application program.

[0009] In order to achieve the above-described technical objects, according to another aspect of the present invention, an apparatus for detecting maliciousness of a non-portable executable file includes an application program running unit which executes a non-portable executable file by running an application program corresponding to the non-portable executable file in a virtual environment; an application program executing flow changing unit which monitors an execution of the application program, breaks the execution of the application program at a predetermined breakpoint during the monitoring of the execution of the application program, changes the executing flow of the application program in a breaking state of the execution of the application program, and resumes the execution of the application program; and a malicious behavior detecting unit which detects a malicious behavior executed after resuming the execution of the application program.

[Advantageous Effects]

[0010] According to the present invention described above, it is possible to effectively detect the maliciousness of a non-portable executable file designed such that when a specific condition such as a version of an application program or an operating system environment is not satisfied, a malicious behavior does not occur.

[0011] Effects of the present invention are not limited to the above-mentioned effects, and other effects, which are not mentioned above, can be clearly understood by those skilled in the art from the following descriptions.

[Description of Drawings]

[0012]

FIG. 1 is a block diagram of an apparatus for detecting a maliciousness of a non-portable executable file according to an exemplary embodiment of the present invention.

FIG. 2 is a flowchart of a method for detecting a maliciousness of a non-portable executable file according to an exemplary embodiment of the present invention.

FIG. 3 illustrates an example of an operation of changing an executing flow by changing a flag during a process of executing a HWP program.

FIG. 4 illustrates a result that a HWP program is continuously executed without being ended so that a malicious behavior is detected.

FIG. 5 illustrates an example of a malicious macro designed to identify feature information of a virtual environment which is being executed so that if it is a virtual environment, the malicious behavior is not conducted.

FIG. 6 illustrates an example of changing a screen size value stored in a register.

FIG. 7 illustrates an example of changing a value indicating whether there is a sound driver stored in a memory.

[Best Mode]

[0013] Hereinafter, exemplary embodiments of the present invention will be described in detail with reference to the drawings. Substantially same components in the following description and the accompanying drawings may be denoted by the same reference numerals and redundant description will be omitted. Further, in the description of the exemplary embodiment, if it is considered that specific description of related known configuration or function may cloud the gist of the present invention, the detailed description thereof will be omitted.

[0014] FIG. 1 is a block diagram of an apparatus for detecting a maliciousness of a non-portable executable file according to an exemplary embodiment of the present invention.

[0015] The apparatus for detecting a maliciousness of a non-portable executable file according to the exemplary embodiment includes a user interface 110, a virtual environment generating unit 120, an application program storing unit 130, an application program running unit 140, an application program executing flow changing unit 150, and a malicious behavior detecting unit 160.

[0016] The user interface 110 provides an interface to select a directory in which a non-portable executable file to be inspected is stored or a non-portable executable file.

[0017] The virtual environment generating unit 120 generates a virtual environment 180 in a computer environment in which the apparatus for detecting a maliciousness of the non-portable executable file according to the exemplary embodiment of the present invention is implemented. For example, the virtual environment 180 may be a well-known sandbox. The virtual environment 180 has a flag representing a process state, a register, and a memory.

[0018] The application program storing unit 130 stores various types of application programs to execute the non-portable executable file to be inspected. The application program storing unit 130 may store application programs such as acrobat reader, MS-word, Power point, Excel, HWP, an image viewer program, a video viewer program, or Internet Explorer.

[0019] The application program driving unit 140 determines a format of the non-portable executable file selected by the user interface 110 and selects an application program corresponding to the format of the non-portable executable file from the application program storing unit 130. The application program running unit 140 runs the selected application program in the virtual environment 180 to execute the non-portable executable file.

[0020] The apparatus for detecting the maliciousness of the non-portable executable file according to the exemplary embodiment of the present disclosure includes the application program executing flow changing unit 150 which detects the maliciousness of the non-portable executable file regardless of a specific condition such as a version of the application program or an operating system environment.

[0021] The malicious non-portable executable file may have branching points that terminates an application program or branches to a flow where no malicious behavior occurs if the specific condition such as the version of the application program or the operating system environment is not satisfied. The non-portable executable file is analyzed in advance by an analyzer to set a breakpoint at the branching point having this possibility. Therefore, a condition which is associated with the branching point to continuously execute the application program without terminating the application program or induce a flow that the malicious behavior occurs may be set.

[0022] The application program executing flow changing unit 150 monitors the execution of the application program and breaks the execution of the application program at the branching point set as a breakpoint during the monitoring of the execution of the application program. The application program executing flow changing unit 150 changes the executing flow of the application program to continuously execute the application program or generate a malicious behavior using the set condition and then resumes the execution of the application program. The application program executing flow changing unit 150 changes a process state, a register value, or a value of a specific address of the memory to change the executing flow of the application program at the branching point.

[0023] The malicious behavior detecting unit 160 monitors an executing process of the application program to detect the malicious behavior through a general behavior based inspecting method. The malicious behavior which is not detected if the executing flow of the application program is not changed appears because the executing flow of the application program is changed by the application program executing flow changing unit 150 so that the malicious behavior is detected by the malicious behavior detecting unit 160.

[0024] FIG. 2 is a flowchart of a method for detecting a maliciousness of a non-portable executable file according to an exemplary embodiment of the present invention.

[0025] In step 210, the application program driving unit 140 runs the application program corresponding to the non-portable executable file in the virtual environment 180 to execute the non-portable executable file.

[0026] In step 220, the application program executing flow changing unit 150 starts monitoring of the execution of the application program.

[0027] In step 250, the application program executing flow changing unit 150 breaks the execution of the application program at the set breakpoint.

[0028] In step 260, the application program executing flow changing unit 150 changes the process state, the register value, or the value of the specific address of the memory so as to change the executing flow of the application program.

[0029] In step 270, the application program executing flow changing unit 150 resumes the execution of the application program and continues to monitor the execution of the application program.

[0030] In the meantime, when the malicious behavior occurs in step 230 while monitoring the execution of the application program, the malicious behavior detecting unit 160 detects the malicious behavior in step 240.

[0031] When the malicious behavior is detected by the malicious behavior detecting unit 160, the apparatus for detecting a maliciousness of the non-portable executable file outputs a detection result that the corresponding non-portable executable file is malicious and ends the detecting process.

[0032] Even though the application program is executed along all executing flows which are branched from all set breakpoints, if the malicious behavior is not detected by the malicious behavior detecting unit 160, the apparatus for detecting the maliciousness of the non-portable executable file outputs the detection result that the corresponding non-portable executable file is not malicious and ends the detecting process.

[0033] An exemplary embodiment in which the application program executing flow changing unit 150 changes the process state to change the executing flow of the application program will be described as follows.

[0034] The process manages the process state by the data calculation during the executing process with information of a combination of switches which are flags. The flags have a value of 0 or 1 to serve as a switch. When the branching point is encountered during the process of executing the process, the branching is performed according to the value (0 or 1) of the switch identified from the instruction. For example, when two values are compared and the values are equal, a Zero Flag (ZF) is set to 1 and when two arbitrary values are added to exceed 4 bytes (based on a 32-bit operating system), Overflow Flag (OF) is set to 1. Thereafter, the executing flow is branched using a condition jump instruction so as to jump if ZF is 0 (JE, Jump if Equal) or jump if ZF is 1 (JNE, Jump if Not Equal).

[0035] FIG. 3 illustrates an example of an operation of changing an executing flow by changing a flag during a process of executing a HWP program. Referring to FIG. 3, in the address 0283C769, a value of a register a1 is compared with 10 and a value of ZF (1 if they are equal, or 0 otherwise) indicating the process state of whether two values are equal is set to 0. In the address 0283C76B, if the previous comparison result is that two values are equal, an instruction to jump to a designated address 283C825 is executed. At this time, the value of ZF is identified to determine whether to jump. Here, if the comparison result is that two values are equal, it jumps to the address 283C825 to continuously execute the application program. However, if the comparison result is that two values are not equal, a next instruction is executed and as a result, the version of the application program does not match so that the application program is forcibly terminated. Accordingly, when the executing flow of the application program is left unchanged, the value of ZF is 0 so that the application program is terminated.

[0036] According to the exemplary embodiment of the present disclosure, a breakpoint is set to the address 0283C76B and the value of ZF is set to be changed to 1. When the value of ZF is changed to 1 in the breaking state and the execution of the application program is resumed, the procedure jumps to the address 283C825 according to the condition jump instruction to continuously execute the application program so that the malicious behavior occurs during the continuous execution of the application program. The malicious behavior is detected by the malicious behavior detecting unit 160. FIG. 4 illustrates a result that a HWP program is continuously executed without being ended so that a malicious behavior is detected. If the value of ZF is not changed, the application program is forcibly terminated so that the malicious behavior is not detected.

[0037] Even though in the present exemplary embodiment, an example that ZF is changed to change the process state is described, not only ZF, but also various flags such as Sign Flag (SF), Overflow Flag (OF), Auxiliary Carry Flag (AC), Carry Flag (CF) may be changed to change the process state.

[0038] According to the exemplary embodiment of the present invention, when the non-portable executable file has a logic that terminates the application program because the condition for operating the malicious behavior (for example, a version of the application program) is not satisfied, the application program is continuously executed by changing the process state at the branching point so that the operation of the malicious behavior may be accurately detected.

[0039] An exemplary embodiment in which the application program executing flow changing unit 150 changes a register value to change the executing flow of the application program will be described as follows.

[0040] A behavior based inspecting method which detects a suspicious behavior by executing the non-portable executable file performs the analysis in an isolated virtual environment to execute the malicious code. The virtual environment has unique feature information and a malicious code in the form of a highly created non-portable executable file may be designed to identify the feature information of the virtual environment to prevent the malicious behavior. For example, when a display size is smaller than a predetermined size, it is designed to be determined as a virtual environment so that the malicious behavior is not conducted or when a memory size is smaller than a predetermined size, it is designed to be determined as a virtual environment so that the malicious behavior is not conducted.

[0041] FIG. 5 illustrates an example of a malicious macro designed that feature information of a virtual environment which is being executed is identified using Excel 4.0 macro so that when the executing environment is not a virtual environment, the malicious behavior is conducted and when the executing environment is a virtual environment, the malicious behavior is not conducted.

[0042] Referring to FIG. 5, instructions of a second cell and a fourth cell are as follows.

```
A2 = GET.WORKSPACE(13)
A4 = IF(A2<770,CLOSE(FALSE),)
```

[0043] The instructions execute the instruction GET.WORKSPACE(13) in the cell A2 to get and store the size (a horizontal size) of the screen and in the cell A4, when the value A2 is smaller than 770, ends the macro by the instruction CLOSE. When the value A2 is not smaller than 770, a next instruction is continuously executed. A screen size of the virtual environment is generally smaller than 770 so that in the virtual environment, the macro ends according to the comparison result in the cell A4. Therefore, the maliciousness is not identified by a behavior based inspecting method of the related art.

[0044] In the exemplary embodiment of the present invention, the breakpoint is set in the address of the branching point corresponding to the cell A4 and the screen size value stored in the register is changed so that the macro is continuously executed without being ended to cause the malicious behavior.

[0045] FIG. 6 illustrates an example of changing a screen size value stored in a register according to the exemplary embodiment of the present invention. The size of the screen taken through the instruction GET.WORKSPACE(13) is stored in the EAX register. Referring to FIG. 6, the screen size 0x380 in the virtual environment which is being executed is stored. Here, when the value of the EAX register is changed to a sufficiently large value, 0x9999, as a comparison result of the cell A4, the value of A2 is larger than 7700 so that a next instruction is continuously executed and the malicious behavior occurs.

[0046] Even though in the present exemplary embodiment, an example that the value of the EAX register is changed to change the executing flow of the application program has been described, not only the EAX register, but also values of various registers such as EBX register, ECX register, EDX register, ESI register, EDI register, EBP register, and ESP register are changed to change the executing flow of the application program.

[0047] According to the exemplary embodiment of the present disclosure, in the case of the non-portable executable file including a malicious macro having a logic which identifies an execution environment to end the macro when the condition is not satisfied, the value stored in the register is changed at the branching point so that the macro is continuously executed without being ended to accurately detect that the malicious behavior is conducted.

[0048] An exemplary embodiment in which the application program executing flow changing unit 150 changes a value of a specific address of a memory to change the executing flow of the application program will be described as follows.

[0049] Referring to FIG. 5 again, the instruction of the first cell is as follows.

A1 = IF(GET.WORKSPACE(42)"CLOSE(TRUE))

[0050] As a result of executing the instruction GET.WORKSPACE(42) to identify whether there is a sound driver, if there is a sound driver, a next instruction is continuously executed and if there is no sound driver, the macro is ended by the instruction CLOSE. In the virtual environment, there is no sound driver so that according to the result of executing the instruction GET.WORKSPACE(42) the macro ends. Accordingly, the behavior based inspecting method of the related art cannot determine the maliciousness.

[0051] In the exemplary embodiment of the present invention, the breakpoint is set in the address of the branching point corresponding to the cell A1 and a value indicating whether there is a sound driver stored in a specific address of the memory is changed so that the macro is continuously executed without being ended to cause the malicious behavior.

[0052] The instruction GET.WORKSPACE(42) gets the corresponding return value to store the return value in the EAX register and performs a predetermined operation (shr eax,1, and eax, 1) on the value stored in the EAX register to indicate the existence of the sound driver as a Boolean value and store the value in a memory address indicated by the EDI register. FIG. 7 illustrates that "0" indicating that there is no sound driver is stored in the address 001335A8 indicated by the EDI register. It is determined that if a value stored in the address 001335A8 is 0, there is no sound driver and if the value is 1, there is a sound driver. When the value of the address 00135A8 indicated by the EDI register is changed from 0 to 1, a next instruction is continuously executed and the malicious behavior may occur.

[0053] Even though in the present exemplary embodiment, an example that the value of the address indicated by the register EDI is changed to change the executing flow of the application program has been described, not only the register EDI, but also values of the addresses indicated by various registers such as the EAX register, the EBX register, the ECX register, the EDX register, the ESI register, the EBP register, and the ESP register are changed to change the executing flow of the application program.

[0054] As described above, according to the exemplary embodiment of the present disclosure, in the case of the non-portable executable file including a malicious macro having a logic which identifies an execution environment to end the macro when the condition is not satisfied, the value stored in the specific address of the memory is changed at the branching point so that the macro is continuously executed without being ended to accurately detect that the malicious behavior is conducted.

[0055] According to the above-described exemplary embodiments of the present disclosure, even in the case of the non-portable executable file designed such that when the execution condition is not satisfied, the malicious behavior is not generated, the process state, the value of the register, or the value stored in the memory is manipulated to cause the malicious behavior.

[0056] The combinations of blocks of the block diagrams and steps in the flowcharts of the present invention may be implemented by computer program instructions. The computer program instructions may be loaded in a processor of a general purpose computer, a special purpose computer, or other programmable data processing apparatus, so that the instructions executed via the processor of the computer or other programmable data processing apparatus create means for implementing the functions described in the blocks of the block diagrams or the steps in the flowcharts. These computer program instructions may also be stored in a computer-usable or computer readable memory that may direct a computer or other programmable data processing apparatus to function in a particular manner, so that the instructions stored in the computer usable or computer readable memory produce a manufacturing article including instruction means which implement the function indicated in the blocks of the block diagrams or the steps in the flowcharts. The computer program instructions may be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions executed on the computer or other programmable apparatus provide steps for implementing the functions described in the blocks of the block diagrams or the steps in the flowcharts.

[0057] Each block or each step may represent a part of a module, a segment or a code, including one or more executable instructions for executing specific logical function(s). In addition, it should be noted that the functions mentioned in the blocks or steps may occur out of order in several alternative embodiments. For example, two blocks or steps shown in succession may be executed substantially concurrently, or may be executed in reverse order according to corresponding functions.

[0058] It will be appreciated that various exemplary embodiments of the present invention have been described herein for purposes of illustration, and that various modifications, changes, and substitutions may be made by those skilled in the art without departing from the scope and spirit of the present invention. Therefore, the exemplary embodiments of the present invention are provided for illustrative purposes only but not intended to limit the technical concept of the present invention. The scope of the technical concept of the present invention is not limited thereto. The protection scope of the present invention should be interpreted based on the following appended claims and it should be appreciated that all technical spirits included within a range equivalent thereto are included in the protection scope of the present invention.

Claims

1. A method for detecting maliciousness of a non-portable executable file, comprising the steps of:

   executing a non-portable executable file by running an application program corresponding to the non-portable executable file in a virtual environment;
   monitoring the execution of the application program;
   breaking the execution of the application program at a predetermined breakpoint during the monitoring of the execution of the application program;
   changing an executing flow of the application program in a breaking state of the execution of the application program and resuming the execution of the application program; and
   detecting a malicious behavior executed after resuming the execution of the application program.

2. The method for detecting maliciousness of a non-portable executable file of claim 1, wherein the breakpoint is set at a branching point.

3. The method for detecting maliciousness of a non-portable executable file of claim 1, wherein the executing flow of the application program is changed by changing a process state.

4. The method for detecting maliciousness of a non-portable executable file of claim 3, wherein the process state is changed by changing a flag indicating the process state.

5. The method for detecting maliciousness of a non-portable executable file of claim 4, wherein the flag includes at least one of Zero Flag (ZF), Sign Flag (SF), Overflow Flag (OF), Auxiliary Carry Flag (AC), and Carry Flag (CF).

6. The method for detecting maliciousness of a non-portable executable file of claim 1, wherein the executing flow of the application program is changed by changing a value of a register.

7. The method for detecting maliciousness of a non-portable executable file of claim 6, wherein the register includes at least one of the EAX register, the EBX register, the ECX register, the EDX register, the ESI register, the EDI register, the EBP register, and the ESP register.

8. The method for detecting maliciousness of a non-portable executable file of claim 1, wherein the execution of the application program is changed by changing a value of a specific address of the memory.

9. The method for detecting maliciousness of a non-portable executable file of claim 8, wherein the specific address of the memory is an address indicated by the EAX register, the EBX register, the ECX register, the EDX register, the ESI register, the EDI register, the EBP register, or the ESP register.

10. An apparatus for detecting maliciousness of a non-portable executable file, comprising:

    an application program running unit which executes a non-portable executable file by running an application program corresponding to the non-portable executable file in a virtual environment;
    an application program executing flow changing unit which monitors an execution of the application program, breaks the execution of the application program at a predetermined breakpoint during the monitoring of the execution of the application program, changes the executing flow of the application program in a breaking state of the execution of the application program, and resumes an execution of the application program; and
    a malicious behavior detecting unit which detects a malicious behavior executed after resuming the execution of the application program.

11. The apparatus for detecting maliciousness of a non-portable executable file of claim 10, wherein the breakpoint is set at a branching point.

12. The apparatus for detecting maliciousness of a non-portable executable file of claim 10, wherein the executing flow of the application program is changed by changing a process state.

13. The apparatus for detecting maliciousness of a non-portable executable file of claim 12, wherein the process state is changed by changing a flag indicating the process state.

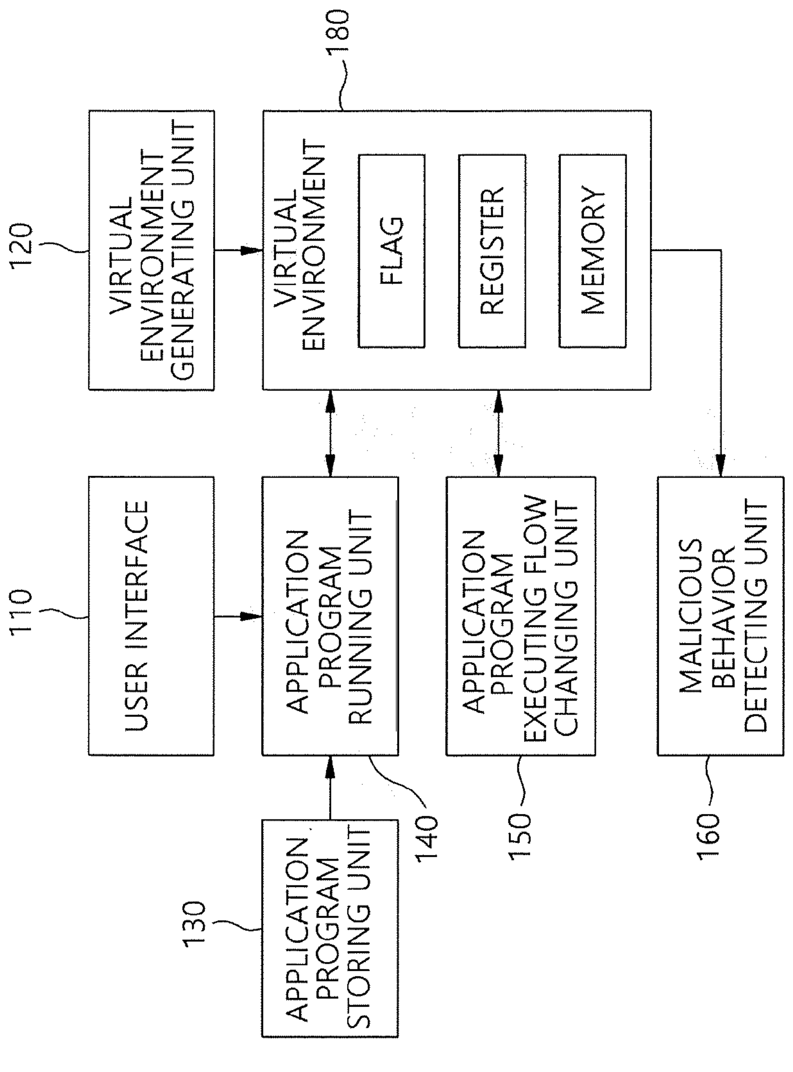
14. The apparatus for detecting maliciousness of a non-portable executable file of claim 13, wherein the flag includes at least one of Zero Flag (ZF), Sign Flag (SF), Overflow Flag (OF), Auxiliary Carry Flag (AC), and Carry Flag (CF).

15. The apparatus for detecting maliciousness of a non-portable executable file of claim 10, wherein the executing flow of the application program is changed by changing a value of a register.

16. The apparatus for detecting maliciousness of a non-portable executable file of claim 15, wherein the register includes at least one of the EAX register, the EBX register, the ECX register, the EDX register, the ESI register, the EDI register, the EBP register, and the ESP register.

17. The apparatus for detecting maliciousness of a non-portable executable file of claim 10, wherein the execution of the application program is changed by changing a value of a specific address of the memory.

18. The apparatus for detecting maliciousness of a non-portable executable file of claim 17, wherein the specific address of the memory is an address indicated by the EAX register, the EBX register, the ECX register, the EDX register, the ESI register, the EDI register, the EBP register, or the ESP register.



110
USER INTERFACE
120
VIRTUAL ENVIRONMENT GENERATING UNIT
180
VIRTUAL ENVIRONMENT
FLAG
REGISTER
MEMORY
130
APPLICATION PROGRAM STORING UNIT
140
APPLICATION PROGRAM RUNNING UNIT
150
APPLICATION PROGRAM EXECUTING FLOW CHANGING UNIT
160
MALICIOUS BEHAVIOR DETECTING UNIT

FIG. 1

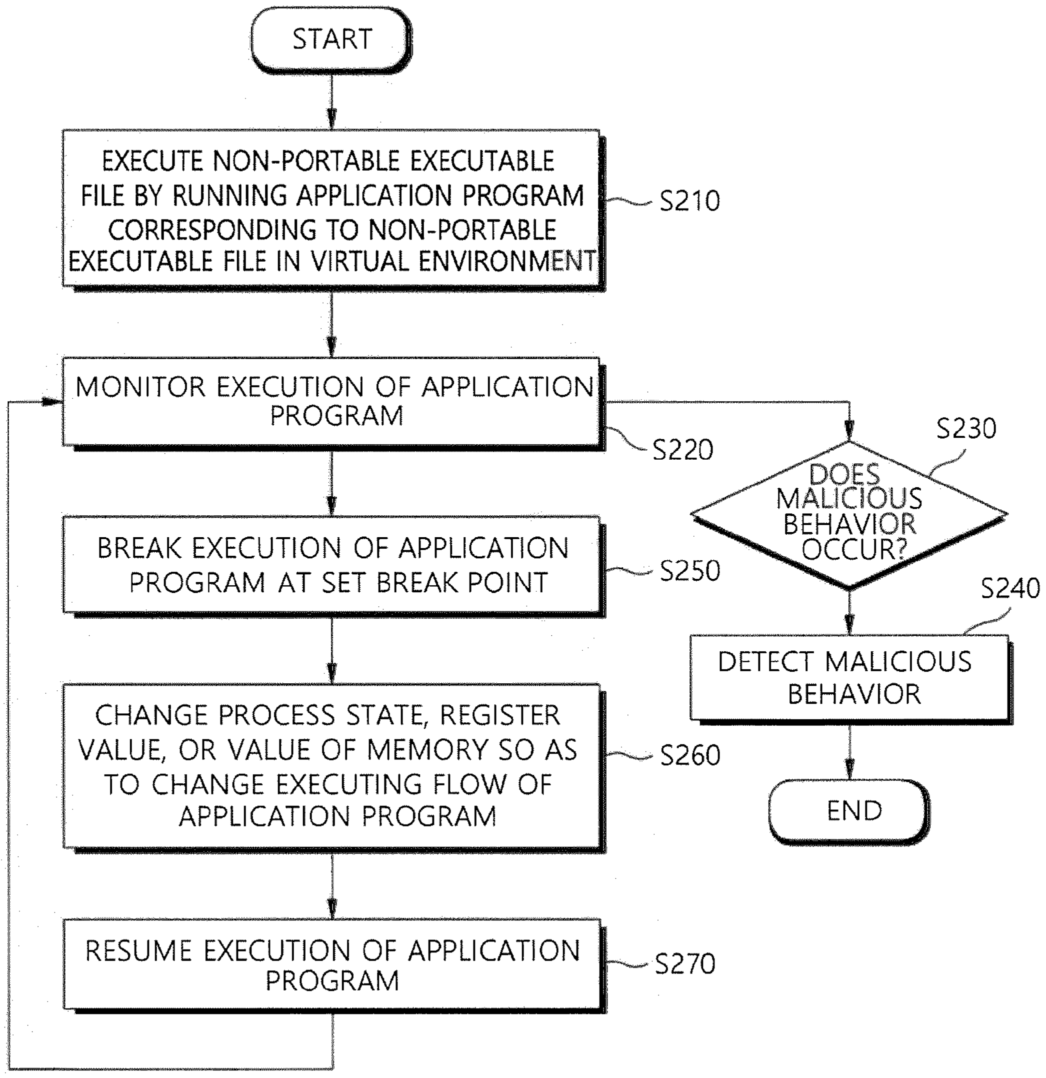
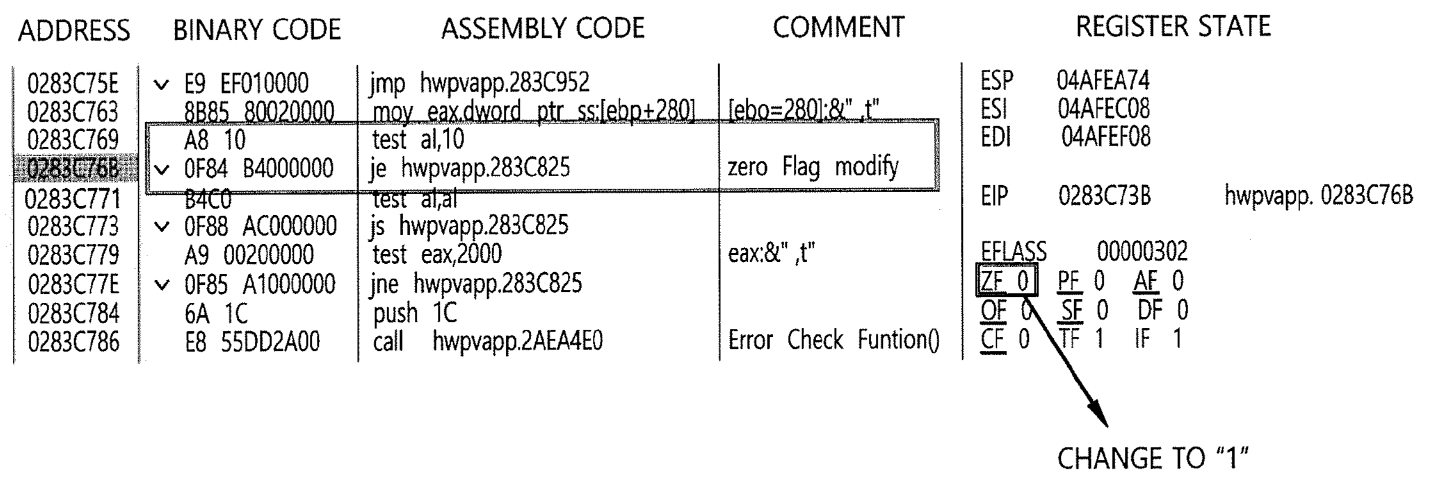
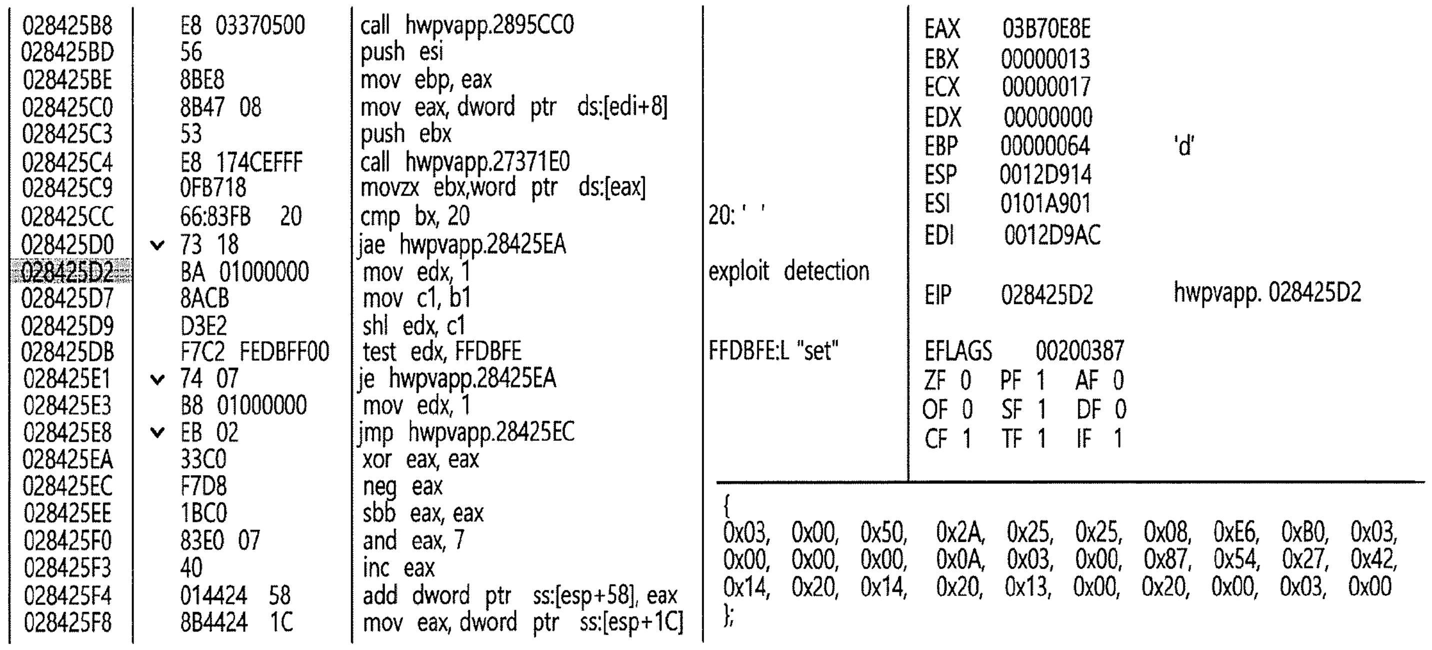

START
EXECUTE NON-PORTABLE EXECUTABLE FILE BY RUNNING APPLICATION PROGRAM CORRESPONDING TO NON-PORTABLE EXECUTABLE FILE IN VIRTUAL ENVIRONMENT
S210
MONITOR EXECUTION OF APPLICATION PROGRAM
S220
DOES MALICIOUS BEHAVIOR OCCUR?
S230
BREAK EXECUTION OF APPLICATION PROGRAM AT SET BREAK POINT
S250
DETECT MALICIOUS BEHAVIOR
S240
CHANGE PROCESS STATE, REGISTER VALUE, OR VALUE OF MEMORY SO AS TO CHANGE EXECUTING FLOW OF APPLICATION PROGRAM
S260
END
RESUME EXECUTION OF APPLICATION PROGRAM
S270

FIG. 2



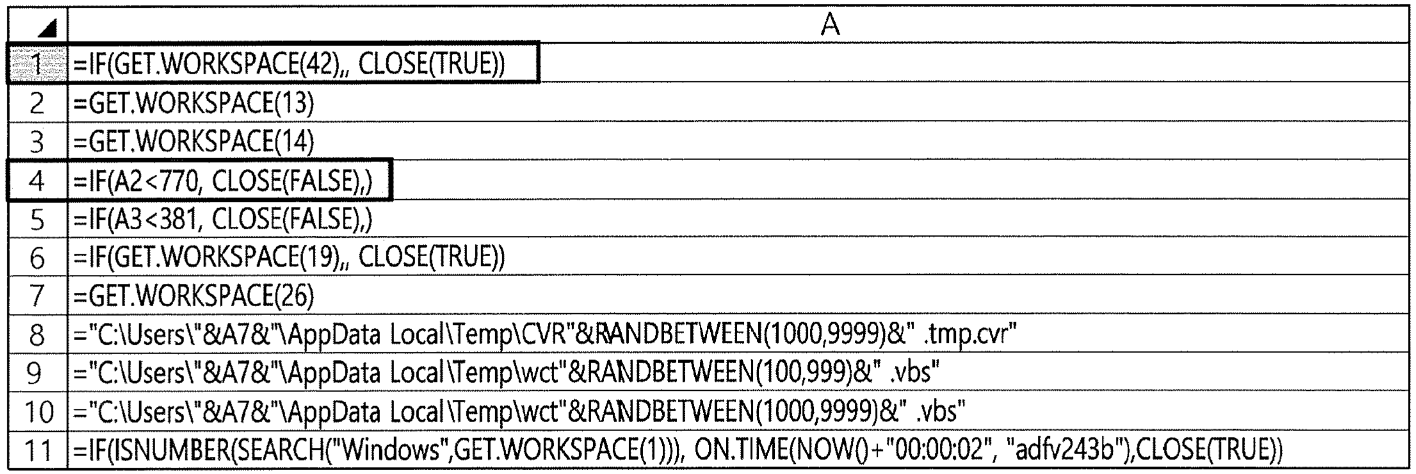
```
ADDRESS    BINARY CODE          ASSEMBLY CODE                          COMMENT                  REGISTER STATE

0283C75E   v E9  EF010000       jmp  hwpvapp.283C952                                            ESP    04AFEA74
0283C763     8B85  80020000     moy  eax,dword  ptr  ss:[ebp+280]     [ebo=280]:&" ,t"         ESI    04AFEC08
0283C769     A8  10             test  al,10                                                     EDI    04AFEF08
0283C76B   v 0F84  B4000000     je  hwpvapp.283C825                    zero  Flag  modify
0283C771     B4C0               test  al,al                                                     EIP    0283C73B        hwpvapp. 0283C76B
0283C773   v 0F88  AC000000     js  hwpvapp.283C825
0283C779     A9  00200000       test  eax,2000                         eax:&" ,t"               EFLASS       00000302
0283C77E   v 0F85  A1000000     jne  hwpvapp.283C825                                            ZF  0   PF  0   AF  0
0283C784     6A  1C             push  1C                                                        OF  0   SF  0   DF  0
0283C786     E8  55DD2A00       call   hwpvapp.2AEA4E0                 Error  Check  Funtion()  CF  0   TF  1   IF  1
```

FIG. 3



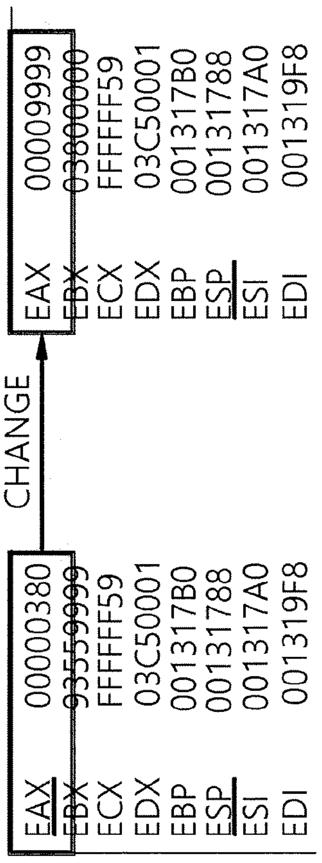
```
028425B8     E8 03370500      call hwpvapp.2895CC0                    EAX     03B70E8E
028425BD     56               push esi                                EBX     00000013
028425BE     8BE8             mov ebp, eax                            ECX     00000017
028425C0     8B47 08          mov eax, dword ptr  ds:[edi+8]          EDX     00000000
028425C3     53               push ebx                                EBP     00000064        'd'
028425C4     E8 174CEFFF      call hwpvapp.27371E0                    ESP     0012D914
028425C9     0FB718           movzx ebx,word ptr  ds:[eax]            ESI     0101A901
028425CC     66:83FB  20      cmp bx, 20               20: ' '        EDI     0012D9AC
028425D0   ˅ 73 18            jae hwpvapp.28425EA
028425D2     BA 01000000      mov edx, 1               exploit detection   EIP  028425D2   hwpvapp. 028425D2
028425D7     8ACB             mov c1, b1
028425D9     D3E2             shl edx, c1
028425DB     F7C2 FEDBFF00    test edx, FFDBFE         FFDBFE:L "set"  EFLAGS    00200387
028425E1   ˅ 74 07            je hwpvapp.28425EA                      ZF 0   PF 1   AF 0
028425E3     B8 01000000      mov edx, 1                              OF 0   SF 1   DF 0
028425E8   ˅ EB 02            jmp hwpvapp.28425EC                     CF 1   TF 1   IF 1
028425EA     33C0             xor eax, eax
028425EC     F7D8             neg eax                  {
028425EE     1BC0             sbb eax, eax             0x03, 0x00, 0x50, 0x2A, 0x25, 0x25, 0x08, 0xE6, 0xB0, 0x03,
028425F0     83E0 07          and eax, 7               0x00, 0x00, 0x00, 0x0A, 0x03, 0x00, 0x87, 0x54, 0x27, 0x42,
028425F3     40               inc eax                  0x14, 0x20, 0x14, 0x20, 0x13, 0x00, 0x20, 0x00, 0x03, 0x00
028425F4     014424   58      add dword ptr  ss:[esp+58], eax      };
028425F8     8B4424   1C      mov eax, dword ptr  ss:[esp+1C]
```

FIG. 4



	A
1	=IF(GET.WORKSPACE(42),, CLOSE(TRUE))
2	=GET.WORKSPACE(13)
3	=GET.WORKSPACE(14)
4	=IF(A2<770, CLOSE(FALSE),)
5	=IF(A3<381, CLOSE(FALSE),)
6	=IF(GET.WORKSPACE(19),, CLOSE(TRUE))
7	=GET.WORKSPACE(26)
8	="C:\Users\"&A7&"\AppData Local\Temp\CVR"&RANDBETWEEN(1000,9999)&" .tmp.cvr"
9	="C:\Users\"&A7&"\AppData Local\Temp\wct"&RANDBETWEEN(100,999)&" .vbs"
10	="C:\Users\"&A7&"\AppData Local\Temp\wct"&RANDBETWEEN(1000,9999)&" .vbs"
11	=IF(ISNUMBER(SEARCH("Windows",GET.WORKSPACE(1))), ON.TIME(NOW()+"00:00:02", "adfv243b"),CLOSE(TRUE))

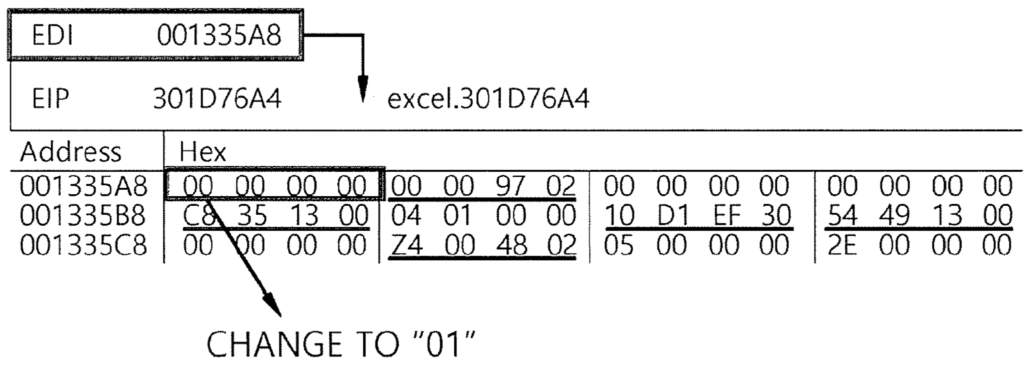
FIG. 5



```
EAX    00000380
EBX    93559999
ECX    FFFFFF59
EDX    03C50001
EBP    001317B0
ESP    00131788
ESI    001317A0
EDI    001319F8
```

```
EAX    00009999
EBX    03800000
ECX    FFFFFF59
EDX    03C50001
EBP    001317B0
ESP    00131788
ESI    001317A0
EDI    001319F8
```

FIG. 6



```
EDI        001335A8
EIP        301D76A4        excel.301D76A4
Address    Hex
001335A8   00 00 00 00 | 00 00 97 02 | 00 00 00 00 | 00 00 00 00
001335B8   C8 35 13 00 | 04 01 00 00 | 10 D1 EF 30 | 54 49 13 00
001335C8   00 00 00 00 | Z4 00 48 02 | 05 00 00 00 | 2E 00 00 00
```

FIG. 7

INTERNATIONAL SEARCH REPORT

International application No.
PCT/KR2021/012194

A. CLASSIFICATION OF SUBJECT MATTER

G06F 21/56(2013.01)i; **G06F 21/53**(2013.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F 21/56(2013.01); G06F 11/00(2006.01); G06F 11/08(2006.01); G06F 11/28(2006.01); G06F 21/00(2006.01); G06F 21/52(2013.01); G06F 21/53(2013.01); G06F 21/55(2013.01); G06F 21/57(2013.01)

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models: IPC as above
Japanese utility models and applications for utility models: IPC as above

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS (KIPO internal) & keywords: 가상 환경(virtual environment), 비실행 파일(non-executable file), 응용프로그램(application program), 모니터링(monitoring), 브레이크(break), 악성(malignant), 탐지(detection)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	KR 10-1646096 B1 (SECULETTER CO., LTD.) 05 August 2016 (2016-08-05) See claim 1.	1-18
Y	KR 10-2009-0065277 A (ELECTRONICS AND TELECOMMUNICATIONS RESEARCH INSTITUTE) 22 June 2009 (2009-06-22) See paragraphs [0018] and [0036]-[0037].	1-18
A	KR 10-1265173 B1 (AHNLAB, INC.) 15 May 2013 (2013-05-15) See paragraphs [0060]-[0098]; and figure 1.	1-18
A	KR 10-2011-0004935 A (ELECTRONICS AND TELECOMMUNICATIONS RESEARCH INSTITUTE) 17 January 2011 (2011-01-17) See paragraphs [0025]-[0042]; and figures 1-2.	1-18
A	KR 10-2004-0098902 A (AHNLAB, INC.) 26 November 2004 (2004-11-26) See paragraphs [0016]-[0039]; and figure 1.	1-18

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"D" document cited by the applicant in the international application
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search **06 April 2022**	Date of mailing of the international search report **06 April 2022**
Name and mailing address of the ISA/KR **Korean Intellectual Property Office** **Government Complex-Daejeon Building 4, 189 Cheongsa-ro, Seo-gu, Daejeon 35208** Facsimile No. **+82-42-481-8578**	Authorized officer Telephone No.

Form PCT/ISA/210 (second sheet) (July 2019)

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/KR2021/012194

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
KR 10-1646096 B1	05 August 2016	None	
KR 10-2009-0065277 A	22 June 2009	KR 10-0926115 B1	11 November 2009
		US 2009-0158260 A1	18 June 2009
		US 8584101 B2	12 November 2013
KR 10-1265173 B1	15 May 2013	JP 2013-239172 A	28 November 2013
		JP 5326062 B1	30 October 2013
		US 2013-0305373 A1	14 November 2013
		US 8627478 B2	07 January 2014
		WO 2013-168913 A1	14 November 2013
KR 10-2011-0004935 A	17 January 2011	KR 10-1060596 B1	31 August 2011
KR 10-2004-0098902 A	26 November 2004	KR 10-0516304 B1	26 September 2005

Form PCT/ISA/210 (patent family annex) (July 2019)